Attorney's Docket No.: 12587-008001 / D01-001

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: INTEGRATING ENTERPRISE SUPPORT SYSTEMS

APPLICANT: LISE WISEMAN AND NICOLE TJON

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL688324259US

I hereby certify under 37 CFR §1.10 that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

August 9, 2001
Date of Deposit

Signature

Gildardo Vargas
Typed or Printed Name of Person Signing Certificate

INTEGRATING ENTERPRISE SUPPORT SYSTEMS

Related Application

[0001] This application claims the benefit of, and incorporates by reference, U.S. Provisional Patent Application No. 60/299,575, filed June 19, 2001.

Background

[0002] This application relates to integrating enterprise support systems such as business support systems (BSSs) and operational support systems (OSSs). A BSS typically is a computer application with which users or other computer processes interact to support normal business functions. BSSs are used across a wide variety of industries including telecommunications, energy, pharmaceutical, government and the like. Examples of BSSs include customer relation management (CRM) applications, billing applications, financial applications, and provisioning applications. OSSs on the other hand relate to the framework of computer software and network architecture underlying the operation and execution of the BSSs. An application for monitoring and/or managing the state of a computer network is one example of an OSS.

[0003] Typically, a single enterprise (e.g., a telecommunications provider) will maintain several BSSs and OSSs (collectively, enterprise applications) that need to share information or otherwise interact. For example, a telecommunications provider

may have a provisioning application for turning on/off switches to control its customers' access to telephone lines or other services, a billing application for automatically generating bills to be sent out to customers, a CRM application for maintaining a database of its customers and for dealing with service calls, complaints and the like, a financial application for handling general accounting functions, and a network management application for managing the underlying network that supports the various enterprise applications.

[0004] Fig. 1 shows a conventional method of integrating multiple enterprise applications. As shown therein, different applications communicate and/or exchange data with one another through specialized point-to-point interfaces (e.g., application program interfaces, or APIs) designed and implemented specifically for certain processes operating within the two applications being connected. Depending on the particular enterprise and the types and number of enterprise applications that it maintains, the number and complexity of point-to-point interfaces that must be designed, implemented and maintained can become significant. For example, the enterprise in Fig. 1 has eleven different applications that require at least 21 separate point-to-point interfaces between processes. In practice, the number of point-to-point interfaces required can greatly exceed the number of enterprise applications because any two applications may require multiple point-to-point interfaces between them – one for each pair of processes that need to communicate.

[0005] In general, implementing and maintaining such specialized point-to-point interfaces is time-consuming and expensive, not only because of the sheer number of point-to-point interfaces that may be required but also due to the complexity and disparity of the applications being connected. For example, different enterprise applications, especially those provided by different manufacturers, may use different programming and/or control languages, may rely on different data models, and generally present several different levels and types of incompatibilities.

Summary

[0006] The present inventors recognized that building and maintaining separate point-to-point interfaces between enterprise applications is cost and time inefficient. Consequently, the present inventors developed various systems and techniques for integrating enterprise applications using an underlying framework of components and tools that dramatically reduce the time and effort required. Implementations of these systems and techniques may include various combinations of the following features.

[0007] In one aspect, facilitating the exchange of information among applications (e.g., business support systems or operational support systems or a combination thereof) may involve receiving a data object from a first application, using a first controller (e.g., a controller class defined in an object-oriented programming language) to route the received data object to a first transformer (e.g., a transformer class defined

in an object-oriented programming language), using the first transformer to transform the data object from a first format used by the first application into a common format object, publishing the common format object to a communication channel, receiving a request from a subscribing application to subscribe to the communication channel, using a second controller to route the common format object to a second transformer, using the second transformer to transform the common format object into a data object in a second format used by the subscribing application, and sending the data object in the second format to the subscribing application.

[0008] The data object received from the first application may correspond to one or more of a plurality of business events. Each controller may correspond to an associated transformer. Each transformer may correspond to a unique transformation from one format to another. Transforming a data object from a format used by the first application into the common format object may involve translating the data object from a vendor-specific format associated with the first application to an Interface Data Language (IDL) object and storing the IDL object in a shared object model. The shared object model may be implemented as a central repository of data objects corresponding to business events. Transforming the data object from a format used by a first application into the common format object may be performed in response to the recognition of a business event by the first application.

[0009] In general, facilitating the exchange of information among applications may

be performed in accordance with a plurality of process models that collectively define when information is to be exchanged among applications. Moreover, if requests are received from a plurality of subscribing applications, then, for each subscribing application, the common format object may be transformed (e.g., using an associated transformer) into a format corresponding to the subscribing application and sent to the subscribing application. Publishing the common format data object to a communication channel may be performed in accordance with a channel architecture that defines a plurality of communication channels having relative priorities. Transforming the common format object into a data object in the second format used by the subscribing application may involve retrieving a stored IDL format object from a central repository and translating the IDL object into a vendor-specific format associated with the subscribing application.

[0010] In another aspect, a system for facilitating the exchange of information among applications may include a plurality of process models, each defining one or more conditions for sending a business event from an application to one or more other applications, a shared object model configured to store data objects received from applications in a common format, a plurality of transformer classes configured to translate data object from a format used by one or more applications into the common format or vice versa, and a plurality of controller classes configured to route data objects to associated transformer classes.

[0011] The system further may include a channel architecture defining a plurality of communication channels, and/or relative priorities for the channels, to which data objects from an application are to be published. The system also may include an acknowledgement class configured to exchange status messages among applications and/or to perform exception handling.

[0012] In various implementations, each process model may correspond to a different business event, the shared object model may include a central repository of data objects in an Interface Description Language (IDL) format, each transformer class may correspond to a unique application format-common format translation, each controller class may be configured to route data objects to an associated transformer class according to a specific process model, and/or the transformer classes and the controller classes may be implemented as classes in an object-oriented programming language such as Java.

[0013] One or more of the following advantages may be provided. The techniques and methods described here result in an end-to-end solution to pre-integrate CRM, billing, provisioning and other BSS and OSS systems. The integration hub is able to integrate virtually any type of application, whether pre-packaged, custom-built, legacy or other dissimilar systems. Accordingly, the integration of disparate systems can be accelerated with corresponding decreases in time, effort, cost and complexity. For example, both the initial development time and costs for integrating systems as well as

the expense required for subsequent improvements and modifications can be reduced dramatically. As a result, an enterprise can easily and quickly become fully integrated to have real time visibility and control of the business and customer experience.

[0014] Moreover, the integration effort can be accelerated because the tasks are not started from scratch. Rather, a shared object model defines the data entities and the corresponding attributes. As a result, the integration effort is cost effective and the long-term costs associated with maintaining BSS and OSS systems can be reduced accordingly.

[0015] The details of one or more embodiments are set forth in the accompanying drawings and the description below. Other features, objects, and advantages will be apparent from the description and drawings, and from the claims.

Drawing Descriptions

[0016] Fig. 1 is a block diagram showing point-to-point interfaces between enterprise applications.

[0017] Fig. 2 is a block diagram showing an example of how the integration hub may be implemented.

[0018] Fig. 2A shows an example definition and structure of a transformer class.

[0019] Figs. 2B, 2C and 2D collectively show an example definition and structure of a controller class.

[0020] Fig. 2E shows an example definition and structure of an acknowledgement class.

[0021] Fig. 3 is a block diagram showing additional details of an example integration hub implementation.

[0022] Figs. 3A and 3B collectively show examples of transformer rules.

[0023] Fig. 4 is a process flow diagram showing an example of a business event being generated and published by one application and subscribed to by another application.

[0024] Figs. 5A-5L are business event definition tables.

[0025] Fig. 6 is a block diagram of a channel architecture.

[0026] Fig. 7 shows a Channel Architecture Definition table.

Detailed Description

[0027] The present inventors have developed a framework, referred to as the "integration hub," for integrating BSS and OSS solutions based on an enterprise's particular objectives. The integration hub is built around enterprise application integration (EAI) middleware (specifically, "BusinessWare" v3.1 available from Vitria, Inc. in Sunnyvale, California) that provides an architectural framework, data requirements, and data conversion rules, which collectively facilitate the integration of "best of breed" enterprise applications such as CRM applications (e.g., Siebel

eCommunications 2000, Clarify CommCenter v3.1), billing (e.g., Portal Infranet v6.1, Lucent Arbor/BP v9.1), and provisioning packaged systems (Architel OMS v1.6.2).

[0028] Fig. 2 is a diagram of an example integration hub implementation. As shown therein, the integration hub 200 uses Vitria to provide message translation 204 and facilitate the exchange of state information 202 among various enterprise applications 207-212 (e.g., order entry 207, order management 208, billing 209, provisioning 210, inventory management 211, fulfillment 212) and further facilitates communication and interaction among the enterprise applications 207-212 and various classes of end-users 213-216 (e.g., partners 213, suppliers 214, internal users 215, customers 216). The integration hub 200 communicates with each of the various entities 207-216 using a "connector" – a multi-layered software communication channel tailored to a specific enterprise application.

[0029] The integration hub is implemented as a distributed collection of software components that interact to enable enterprise applications to seamlessly and easily share information in the form of "business events." A business event represents an abstract function or activity that is common to many business systems, for example, create account, create order, cancel order, generate invoice, etc. The integration hub facilitates exchange of business events among disparate enterprise applications, for example, by translating, or transforming, business events from one proprietary format into another, via a common, shared format. There are six primary building blocks used

in implementing an integration hub instance: process models, a shared object model (SOM), controller classes, transformer classes, acknowledgement class, and the channel architecture, each of which is described below.

[0030] • Process Model – A process model defines a process of when to send a business event from a first enterprise application (e.g., a provisioning system) to one or more other enterprise applications (e.g., a billing system) and further defines what information is needed for each system. A separate process model is defined for each different type of business event used in the integration hub implementation. For example, depending on the specific enterprise applications involved and the particular objectives of the enterprise, a "cancel service" process model could be defined that upon detecting the creation of a "cancel service" business event in the CRM system will direct that that business event also be transformed and sent to the provisioning system (e.g., to turn off a telephone line) and to the billing system (e.g., to terminate further charges and/or generate a final invoice). A process model typically includes, at a minimum, the following components: a model server, model properties, business event definitions, business process object attribute definitions, and business process states, transitions and actions.

[0031] • Shared Object Model (SOM) – The SOM is a central repository that represents enterprise data objects in common format – namely, the Interface Description Language (IDL) format. Business events received from an enterprise application in a proprietary format are transformed into the SOM format, and then when needed by another enterprise application, are transformed from the SOM format into the proprietary format of the requesting application. A SOM typically includes, at a minimum, a list of IDL structures and structure members, a list of arrays (including type and size), a list of application values and IDL values, a mapping of values to IDL fields, a mapping of IDL structures to Business Events, and a mapping of Interface Events to Business Events.

[0032] • Transformer Classes (or "transformers") – The transformer classes are Java-language classes that translate data (e.g., business events) from enterprise application format to SOM format, and/or from SOM format to enterprise application format. Fig. 2A shows an example of the definition and structure of a transformer class, "SlToVtAddProductTransformer." A separate transformer class exists for each different transformation that may need to be performed. For example, exchanging a business event

between a Siebel eCommunications CRM system and a Portal Infranet billing system would require four separate transformer classes: (1) a transformer class to translate from Siebel eCommunications format to SOM format; (2) a transformer class to translate from SOM format to Siebel eCommunications format; (3) a transformer class to translate from SOM format to Portal Infranet format; (4) a transformer class to translate from Portal Infranet format to SOM format. For example, in the embodiment of Fig. 3 (which shows an Integration Hub integration of four different enterprise applications), the following transformer classes may be used: SIToVtAddProductTransformer, SIToVtAddServiceTransformer, SIToVtAppAccLvlAdjTransformer, SIToVtCancelProductTransformer, SIToVtCancelServiceTransformer, SIToVtCreateAccountTransformer, SIToVtModifyAccountTransformer, SIToVtModifyServiceTransformer, SIToVtOrderHeaderTransformer, SIToVtUpdAccStatusTransformer, SLTransformer, VtToSIUpdPrdctStatusTransformer, VtToSIUpdSrvcStatusTransformer, PITransformer, VtToPIAcctAdjustmentTransformer, VtToPIAddProductTransformer, VtToPIAddServiceTransformer, VtToPICancelProductTransformer, VtToPICancelServiceTransformer, VtToPICreateAccountTransformer, VtToPIModifyAccountTransformer, VtToPIModifyServiceTransformer,

VtToPlUpdateAcctStatusTransformer, ABPTransformer, VtToABPAddProductTransformer, VtToABPAddServiceTransformer, VtToABPApplyAcctLvlAdjmntTransformer, VtToABPCancelProductTransformer, VtToABPCancelServiceTransformer, VtToABPCreateAccountTransformer, VtToABPModifyAccountTransformer, VtToABPModifyServiceTransformer, VtToABPUpdateAccountStTransformer, OMSTransformer, OMSToVtProvisioningResponseTransformer, OMSToVtServiceStatusNotification, and VtToOMSRequestBroadbandTransformer.

[0033] • Controller Classes (or "controllers") – The controller classes are Java-language classes that route business events to appropriate transformer classes. For example, if the process model specified that a business event generated at the Siebel eCommunications CRM system needs to be sent to the Portal Infranet billing system, one controller class would route the Siebel eCommunications-formatted business event to the transformer that translates from Siebel eCommunications format to SOM format, and another controller class would route the SOM-formatted business event to

the transformer that translates from SOM format to Portal Infranet format. Figs. 2B, 2C and 2D collectively show an example of the definition and structure of a controller class, "PIController." In the embodiment of Fig. 3, for example, the following controller classes may be used: SIController, PIController, ABPController, and OMSController.

[0034] • Acknowledgement Class – The acknowledgement class is a Java-language class that is used to send status messages from an enterprise application to one or more other enterprise applications and/or for exception handling. For example, each time a business event is received by an enterprise application, the receiving application invokes an acknowledgement class to signal either that the transfer was successful or that a particular error occurred. Upon occurrence of error, the acknowledgement class can trigger an appropriate error-handling procedure. Fig. 2E shows an example of the definition and structure of an acknowledgement class, "SIAcknowledgementTransformer."

[0035] • Channel Architecture – The channel architecture defines the communication channels to which business events will be published and the relative priority of those channels. For example, the following

channels are used in the integration hub implementation depicted in Fig. 3: AccountChannel, ProvisionOrderChannel, ToSiebel, Acknowledgement Channel, ProvisioningRequest, ProvisioningResponse, BillingChannel, ToArbor, and ToPortal. Each of the integration hub business events is assigned a specific channel. Further details on the Channel Architecture are provided below.

[0036] Fig. 3 shows details of an example integration hub implementation. In this example, four different enterprise applications are integrated using the integration hub 300: a Portal Infranet billing application 302, an Architel provisioning application 304, and a Siebel eCommunications CRM application 306 and an Arbor billing application 308. The integration hub 300 is comprised of a shared object model (SOM) 312, a communicator infrastructure 310, a channel architecture 314 and connectors 316, 317. The SOM 312 provides a predefined common logical model of business objects and events specific to a particular industry.

[0037] The SOM 312 is an IDL representation (defined using the Vitria software developers' kit (SDK)) of common business entities in Vitria BusinessWare. These business entities include account, customer, contact, order, payment, and billing information. The integration hub IDL defines the data format, modules, and structures of common business entities such as account, product, service, order, customer,

payment, and contact. Enterprise applications generate functions that are translated into business events within Vitria. Vitria encapsulates the business data for these events, e.g. Create Customer, Add Product, Cancel Service, in an IDL format that the integration hub defines as the shared object model (SOM).

[0038] The SOM 312 is used by the transformation classes, which as described above are a set of java-language classes that use transformation rules 313 to translate data from enterprise application format to SOM format or vice versa. In effect, the transformation rules 313 are business rules that translate from the specific APIs (application program interfaces) used by the enterprise applications into the common SOM format and vice versa. More particularly, the event transformation rules 313 are defined in the transformer classes. These java transformer classes load enterprise application data into Vitria business event objects. The transformers also extract data from business event objects and load them into enterprise application databases or APIs. The transformer classes operate by applying formatting and translation logic to the data before loading them into an application or business event. These translation rules include changing numeric fields to alpha numeric, changing status values to enumerator or Boolean values, and copying string or numeric fields to target string or numeric fields. The transformers include exception handling and acknowledgement classes that capture errors and publish them to an acknowledgement channel or error log.

[0039] The communicator infrastructure 310 provides transactional synchronous and/or asynchronous message structure for communicating between the integration hub 300 and the enterprise applications. It reliably transports information in a consistent format between systems by employing an event-driven publish-subscribe methodology via communications channels. The communicator 310 includes a number of channels to which an application can publish an event and to which another application can subscribe to receive the event. The communicator infrastructure used in this embodiment was developed by, and is available from, Vitria, Inc. of Sunnyvale, California. Further details on the communicator infrastructure can be found in documentation available from Vitria and in information at Vitria's website:

http://www.vitria.com/products/businessware/eai.html

[0040] The channel architecture 314 defines how messaging will be partitioned for high-performance communications. The channel architecture defines the communication channels that business events will be published to. These channels include New Order, Account, and Product. Each of the integration hub business events is assigned a specific channel. The integration hub channels are prioritized (using the priority scheme developed by Vitria) to ensure that business events are sent to enterprise applications in the correct order; e.g. Create Customer is sent to billing before Add Product. Detailed documentation on the priority scheme is available directly from Vitria for customers and integration partners.

[0041] The connectors 316, 317 are part of the communicator 310 and represent the channels for communicating between the integration hub 300 and the enterprise applications. The communicator has two different types of connectors to communicate with each separate application: a publisher connector 317, which publishes a business event from a publishing application to a channel in the communicator 310, and a subscribing connector 316 which receives a business event from the communicator 310 (pushed by a publisher connector 317) into the subscribing application connector.

[0042] Each business event represents a logical business function. Sample business events include account creation, product creation, service modification, and product cancellation. The Create Customer business event, for example, originates in the CRM application, e.g. Siebel or Clarify. The CRM application invokes an API or changes database tables that the integration hub monitors. The integration hub uses its connector architecture, which includes controllers and transformers, to translate the enterprise data into a SOM format within a business event. The integration hub publishes the business event onto the appropriate channel. Subscribers of the event – e.g., billing, provisioning, process automation – subscribe to the business event from the channel and into their respective connectors. The connectors translate the SOM data into the target enterprise application's data format; e.g. Portal Infranet, Architel OMS. The connector invokes an application-specific API to load the data into the target enterprise application. The target enterprise application generates an

Acknowledgement business event in the integration hub that includes the success/failure status and result data of the target enterprise application's API. This Acknowledgement business event is published to the Acknowledgement channel for transmission to another enterprise application or a log file.

[0043] Each Vitria connector 316, 317 is comprised of three different components: a publisher / subscriber module 318 (depending on whether the connector is a publisher connector 317 or a subscriber connector 316), a transformer 320 and a driver 322. The Vitria publisher / subscriber modules 318 send / retrieve business events to / from the communicator infrastructure 310. The Vitria drivers 322 provide pre-built interfaces to packaged applications. As with the communicator infrastructure 310, the Vitria drivers 322 and the Vitria subscriber / publisher modules 318 used in this embodiment were developed by and available from Vitria, Inc., although in other embodiments they could be custom-developed if desired. Further details on the publisher / subscriber module 318 and the driver 322 are provided in documentation available from Vitria.

[0044] The transformer component 320 of each connector 316, 317 provides the execution environment for the event transformation rules 313. More particularly, each transformer 320 is a separate Java-language class that, when invoked, performs a translation function on data (e.g., a business event) using methods defined by the transformation rules. A transformer rule is an algorithm that converts between the message formats to disparate applications. A transformation rule lists the fields of the

related messages of the applications and describes how each field is related to another. Figs. 3A and 3B collectively show an example of the transformation rules used for the CRM, Billing, and middleware applications.

[0045] In general, the integration hub provides a framework that enables a first application, upon having a business event created or otherwise occur locally (such as the creation of a new customer account by a human user), to publish the event to an appropriate channel in the communicator infrastructure, and concurrently have the event converted to a common format (i.e., the SOM format). At that point, any other enterprise application connected to the integration hub can subscribe to the channel of interest and retrieve the newly created event. As part of the retrieval process, the event is converted from the common SOM format into the APIs and fields used by the subscribing application. The subscribing application then can use the APIs and fields to undertake its own local action responsive to, or otherwise appropriate for, the business event generated by the first application.

[0046] Fig. 4 shows an example of data flow during a typical sequence using the integration hub architecture to exchange business events between two applications – in this example, a Siebel eCommunications CRM application and a Portal Infranet billing application. Assume, for example, that a human operator creates a new customer account and a service order for that account in the Siebel eCommunications CRM application running on node 400. Following the submission of the service order, the

CRM Application causes the new account information to be sent in Step 1 to the server 402 which hosts the application's database engine. The Siebel Workflow Manager (a component of the Siebel application) is what performs this send. It is invoked to send customer information to the Integration Hub. Upon submission of the service order, the Siebel Workflow Manager determines that a new account has been created and needs to be sent to the Integration Hub. The Siebel Workflow Manager then goes to the Siebel database and retrieves an instance of a pre-defined Siebel integration object called "Vitria Account". The integration object tells the workflow manager what data to get and where to get it from, for that particular business event. Within Siebel eCommunications, the customer data, which comprises the integration object instance, is transformed into XML (extensible markup language) format and sent over an HTTP (hypertext transfer protocol) protocol to Vitria servlet 403. The Siebel source connector model 407 is a publisher to the Vitria servlet.

[0047] Next, in Step 3, the Siebel source connector model 407 converts the XML data into an IDL format and then a transformer translates the Siebel IDL format into a shared object model format. The source connector 407 in Step 4 creates a "Create Customer" event (in the SOM format) and publishes it onto the channel 409, which receives Create Customer business events that billing applications subscribe to. The integration hub channels are prioritized using Vitria's configuration tools to ensure

business events are received in the correct order; e.g. a Create Customer business event is processed before a Add Product business event.

[0048] Next, in Step 5, the Portal Infranet target connector 410 executing on node 414 subscribes to the channel 409. Next, in Step 6, a transformer transforms the "Create Customer" event from the SOM format into appropriate APIs and field lists for the Portal Infranet application. In Step 7, a Vitria target driver sends the APIs and field lists to the connection manager component 413 of the Portal Infranet application. Finally, in Step 8, Portal Infranet creates a new customer account within its local Account table in its database 415.

[0049] The integration hub's shared object model has defined, and uses, a set of core business events to pass customer, product and service data among the various enterprises applications. The number and types of business events used for a given integration depend on the applications being integrated and the objectives of the enterprise and its users. Often, the business event definitions are industry-specific, for example, specific to the telecommunications industry, because companies in a given industry often use the same or similar enterprise applications in furthering their commercial endeavors.

[0050] For the example of Fig. 5, in which a CRM system and a billing system are integrated to share information with each other, the following ten core business events were defined and used:

- Create Order – starts the provisioning and billing process of an order within the process models
- Add Product – add a new product to an account or service from CRM into provisioning and billing
- Add Service Instance – add a new service instance, service location, login, and password to an account from CRM into provisioning and billing
- Apply Account Level Adjustment – apply account-level adjustments (credits and debits) from CRM into billing
- Cancel Product – cancel an existing product from an account or service from CRM into provisioning and billing
- Cancel Service Instance – cancel an existing service instance from an account from CRM into provisioning and billing
- Create Customer – submission of billing, payment, and contact information for a new account from CRM into billing
- Maintain Account – submission of changed billing, payment, and contact info for an existing account from CRM into billing
- Maintain Service Instance – make corrections to an existing service's location and password from CRM into provisioning and billing
- Update Account Status – transfer changes to account status (active, suspend, closed) from CRM to billing

- Update Product Status – as provisioning system (OMS) passes through its processes, the product status will be updated automatically in Siebel the billing system to the CRM system

[0051] In effect, these 10 business events define a common language for communicating between the Siebel eCommunications 2000 CRM , Portal Infranet v6.1 billing, Lucent Arbor/BP v9.1 billing, and Architel OMS provisioning. The integration hub effectively acts as the intermediary and interpreter to facilitate communication between these systems. Integrating additional applications into the enterprise would generally require the definition and use of different or additional business events specific to the applications under consideration. However, by defining business events that have general applicability to several different types of applications (e.g., Create Customer), business events can be re-used thus reducing the integration effort, expense and complexity. In general, other embodiments could use additional or different business events, depending on the applications being integrated as well as the objectives of the enterprise and its users.

[0052] Figs. 5A-5L are examples of business event definition tables. These particular business events were defined for integrating a Siebel CRM system with two different billing systems: Portal Infranet and Arbor. The set of 12 business events defined below and in Figs. 5A-5L are sufficient to enable the Siebel CRM system, the

Portal Infranet billing system and the Arbor billing system to exchange information that implements the integration objectives of the enterprise. The integration hub defines process models within Vitria that subscribe to the Create Customer, Modify Customer, Update Account Status, Add Product, Add Service, Modify Service, Cancel Service, and Cancel Product business events. The order management process models created in the integration hub define the process management of accounts, orders and services. These process models define the process of when to send business events to the provisioning and billing systems and define what information is needed for each system. As an example, the Add Service Instance and Add Product business events for ordering a telecom service (e.g., a digital subscriber line (DSL)) must first be sent to Architel OMS for provisioning and then to Infranet or Arbor for billing after it has been provisioned. The Add Product for a DSL modem can be sent directly to Infranet or Arbor for billing. For both examples, the process model is a subscriber and publisher of the Add Product business event. The process model subscribes to the Add Product business event after it is published by the CRM, e.g. Siebel, Clarify. As discussed below, the process model, in turn, publishes the Add Product business event to the ProvisioningRequest channel in Fig. 6 and/or the Billing channel in Fig. 6 based on the product type.

[0053] Fig. 5A shows the definition table for the "Acknowledgement" business event, further details of which are set forth below.

File Name:	Acknowledgment Event
Type:	Business Event Definition
Publisher(s):	Billing (Arbor, Portal)
	CRM (Siebel)
Subscriber(s):	Install Service Process Model
	Modify Service Process Model
	Disconnect Service Process Model
	AcknowledgementToFile
IDL File Name:	ihBusinessEvents.idl

Business Event Name: Acknowledgment

Description: An acknowledgement event can be created in two main ways:

1. If the translation or the transformation fails in the Arbor, Portal and Siebel connectors an error message is written to the acknowledgement log file.

2. As Arbor, Portal and Siebel target connectors' process each Business Event, an acknowledgement event will be created from the response returned from the application. This acknowledgement will act as a confirmation to the Vitria process models that Arbor, Portal or Siebel has successfully processed each business event.

[0054] Fig. 5B shows the definition table for the "Add Product" business event, further details of which are set forth below.

File Name:	Add Product Event
Type:	Business Event Definition
Publisher(s):	CRM (Siebel)
	Install Service Process Model
	Modify Service Process Model
	Disconnect Service Process Model
Subscriber(s):	Billing (Portal, Arbor)
	Order Pre-Processor Model
IDL File Name:	ihBusinessEvents

Business Event Name: Add Product

Description: When a new service order containing products that are purchased by a customer, an Add Product event is kicked off. This event passes the deal (i.e. component or bundle) name stored in the CRM as well as the name of the account that it should be added to. This information is sent to the provisioning system if the product requires provisioning. Once provisioning has been completed, the add product event will be sent to the billing system and the product is added to the appropriate account.

[0055] Fig. 5C shows the definition table for the "Add Service" business event, further details of which are set forth below.

File Name:	Add Service Event
Type:	Business Event Definition
Publisher(s):	CRM (Siebel)
	Install Service Process Model
	Modify Service Process Model
	Disconnect Service Process Model
Subscriber(s):	Billing (Arbor, Portal)
	Order Pre-Processor Model
IDL File Name:	ihBusinessEvents

Business Event Name: Add Service

Description: The Add Service event occurs when a new service is ordered. When a new service is ordered, the CRM application passes the details to the billing application i.e. account number, service type, service instance id, password, effective date, and service location via Vitria IOM. Vitria IOM will utilize the service information to create a provisioningRequest event to provision the products associated with the given addService event. Once a product underneath the specific serviceInstance is complete, IOM will then send the addService event to the billing application and create a service instance under the appropriate billing account.

[0056] Fig. 5D shows the definition table for the "Apply Account Level Adjustment" business event, further details of which are set forth below.

File Name:	Apply Account Level Adjustment
Type:	Business Event Definition
Publisher(s):	CRM (Siebel), Account Process Model
Subscriber(s):	Billing (Portal, Arbor) Account Process Model
IDL File Name:	ihBusinessEvents

Business Event Name: Apply Account Level Adjustment

Description: When an adjustment is applied to an account, the adjustment information will be sent from CRM to Billing. Account number, adjustment amount, and reason description will be passed from CRM to Vitria into Billing.

[0057] Fig. 5E shows the definition table for the "Cancel Product" business event, further details of which are set forth below.

File Name:	Cancel Product (Version 1.0)
Type:	Business Event Definition
Publisher(s):	CRM (Siebel)
Subscriber(s):	Billing (Portal, Arbor)
IDL File Name:	ihBusinessEvents

Business Event Name: Cancel Product

Description: This event removes product from an account or service object. This event passes the information such as, deal or component name, product instance id, account number, and optionally, service type, service id (Portal login), and service password from the CRM to Billing through the Vitria connectors and Vitria IOM component. This information is sent to the billing system and a confirmation of delivery is then sent to an acknowledgement log file.

[0058] Fig. 5F shows the definition table for the "Cancel Service" business event, further details of which are set forth below.

File Name: Cancel Service (Version 1.0)

Type:	Business Event Definition
Publisher(s):	CRM (Siebel)
Subscriber(s):	Billing (Portal, Arbor), Provisioning
IDL File Name:	ihBusinessEvents

Business Event Name: Cancel Service

Description: The Cancel Service event cancels a service by changing its status from active to closed. The CRM sends the Cancel Service event (Including account number, service type, service ID, effective date, and status) to the Vitria connector after the service has been cancelled in the CRM.

Upon receipt of the Cancel Service event, Vitria IOM sends a Provisioning Request event (CreateOrder.Request – OMS System Event) to Provisioning and awaits a Provisioning Response event (CreateOrder.Response –OMS System Event). Once provisioning is complete, Vitria IOM sends a Cancel Service event to billing.

Billing receives the account number through the Vitria connector and it uses the account number to find the account. The service POID is found in Portal and the service status is changed by invoking the ChangeStatus opcode. Within Arbor, the service instance id is used to call the cease function on the specific instance of the Service Instance object that needs to be cancelled. Only the service instance id (external_id) and effective date will need to be passed in order to cancel service within Arbor.

[0059] Fig. 5G shows the definition table for the "Create Customer" business event, further details of which are set forth below.

File Name:	Create Customer Event Definition
Type:	Business Event Definition
Publisher(s):	CRM (Siebel) Install Service Process Model Modify Service Process Model Disconnect Service Process Model
Subscriber(s):	Billing (Portal, Arbor) Account Process Model

IDL File Name: ihBusinessEvents

Business Event Name: Create Customer

Description: The Create Customer event occurs when an order for a new customer is submitted. When the first order for a new customer is submitted, the CRM application passes the createAccount event into Vitria and to the Account Process Model. The Account Process Model will then send the createAccount event containing contact, payment, and billing information into the Billing Application. The Billing Application takes this information and creates an account. All optional fields that were not passed from CRM application will have default values in the Billing application. The Billing application will send to Vitria a confirmation indicator message stating whether the Billing application has successfully created the customer or not.

[0060] Fig. 5H shows the definition table for the "Modify Account" business event, further details of which are set forth below.

File Name:	Modify Account
Type:	Business Event Definition
Publisher(s):	CRM (Siebel)
	Install Service Process Model
	Modify Service Process Model
	Disconnect Service Process Model
Subscriber(s):	Billing (Portal, Arbor)
	Account Process Model
IDL File Name:	ihBusinessEvents

Business Event Name: Modify Account

Description: The Modify Account event occurs when a customer's attributes change. Account modifications include changes to an account's billing address, contact information, bill payment method, and bill cycle date. When an account's attributes change, the CRM tool passes the billing system information regarding changes to the customer account. The billing system takes the information and updates the customer's attributes. The billing application will send Vitria an acknowledgement

message when the billing system has successfully updated the customer's information. Examples of account attribute changes are changes to: the account's name, the account's address, the account's contact information, the billing cycle, the billing currency, the bill payment type, and the account close day of month. Note that the currency cannot be changed for any account. The bill frequency, bill type, and bill cycle day can only be modified for non-subordinate accounts.

[0061] Fig. 5I shows the definition table for the "Modify Service" business event, further details of which are set forth below.

File Name:	Modify Service
Type:	Business Event Definition
Publisher(s):	CRM (Siebel) Install Service Process Model Modify Service Process Model Disconnect Service Process Model
Subscriber(s):	Billing (Portal, Arbor) Order Pre-Processor Model
IDL File Name:	ihBusinessEvents

Business Event Name: Modify Service

Description: The Modify Service event will take place in the CRM when any service information is modified.

The information regarding the change will be sent to the Billing Application. There are three alternate scenarios that result in a modifyService event.

1. A new product is added to an existing Service.
2. A product is cancelled from an existing Service, however there are still other active products remaining under the same service.
3. A product is added and a product is cancelled under an existing service.

[0062] Fig. 5J shows the definition table for the "Service Status Notification Error" business event, further details of which are set forth below.

File Name:	Service Status Notification
Type:	Business Event Definition
Publisher(s):	Provisioning (OMS)
	Product Service Model
Subscriber(s):	Install Service Process Model
	Modify Service Process Model
	Disconnect Service Process Model
IDL File Name:	ihBusinessEvents

Business Event Name: Service Status Notification

Description: The purpose of this event is to send provisioning status information from OMS to the Vitria IOM.

When a new service or product request is made (Provisioning Request Event), it is sent to OMS via the OMS Target Connector. A response (Provisioning Response Event) is sent by OMS to Vitria IOM via the OMS Target Connector. The OMS Source Connector will monitor the OMS database to see if there is any change in the status of a service or product. If there is a change in the status of a service or product, a Service Status Notification Event is created with the required parameters. The OMS server is accessed to obtain additional information. The Service Status Notification Event is then sent to Vitria IOM. The Vitria IOM will create and send the Update Product Status Business Event to Siebel (where the CRM information will be updated). Once provisioning is complete, the automator will then trigger the billing events to Portal/Arbor (where the BILLING information will be updated).

[0063] Fig. 5K shows the definition table for the "Update Account Status" business event, further details of which are set forth below.

File Name:	Update Account Status
Type:	Business Event Definition
Publisher(s):	CRM (Siebel)

	Install Service Process Model
	Modify Service Process Model
	Disconnect Service Process Model
Subscriber(s):	Billing (Portal, Arbor)
	Account Process Model
IDL File Name:	ihBusinessEvents

Business Event Name: Update Account Status

Description: The purpose of this event is to send information from the CRM to the Billing application when an account is suspended, re-activated or closed. The CRM sends the account ID and status to Billing through the Vitria connectors and the Account Process Model. When an account status is changed to "Suspended", "Active" or "Closed", the child account's status must be changed to correspond to the same status. The CRM application will change the child services or products to the "Suspended", "Active", or "Closed" status before the account status is changed. This is a user training issues, and must be incorporated in the training plan.

[0064] Fig. 5L shows the definition table for the "Update Product Status" business event, further details of which are set forth below.

File Name:	Update Product Status
Type:	Business Event Definition
Publisher(s):	Provisioning (Architel), Vitria IOM
Subscriber(s):	Vitria IOM, CRM (Siebel)
IDL File Name:	ihBusinessEvents

Business Event Name: Update Product Status

Description: When a product event is triggered it will be sent from Siebel to the Service Level Model. The Service Level Model will then send a provisioning request to the provisioning system (OMS). When the provisioning system receives the request a response will be sent back to the Service Level Model indicating that the request has been received. Siebel will then be notified in order to update the product status. Upon the completion of provisioning all the products OMS will send a Service Status Notification back to the Service Level Model. This will then be passed

onto Siebel and the status of the product will be changed to complete or cancelled depending on the event that triggers this process.

[0065] Fig. 6 is a block diagram of the integration hub's channel architecture 600. To aid in understanding the data flow involved in Fig. 6, an explanation for each component in this figure is provided. The Siebel application 602 is the CRM application where customer data originates. There are specific data fields required by the downstream billing systems, Portal Infranet 604 and Arbor 606. A "channel" is a key organizational concept in the publish/subscribe model in that it accepts incoming events from publishing applications and communicates these events to subscribers of that channel. Fig. 7 is a Channel Architecture Definition table that includes a description of each channel and the associated publisher / subscriber information.

[0066] The Automator Process Model 608 is a Vitria graphical model that specifies application integration logic. Specifically, the automator 608 is the business automation component that controls the flow of data between systems, and ensures the timely delivery of accurate data. The OMS application 610 is used for provisioning purposes. The Acknowledgement channel 612 handles the exception logic by recording processing errors and success messages in a text file (Acknowledgement.txt). The error information may include the name of the failed event, the name of the application for which the connector belongs, the error message, and a flag to denote a success or failure. This information is packaged within an event that is sent to the

Acknowledgement channel 612. Business Events, as described above, are represented in Fig. 6 at the point of each event flow.

[0067] The following example represents an example of data flow that may take place in the architecture 600 when a new customer orders a DSL product. The Required information will be entered into the Siebel system 602 and once the 'Submit' button is pushed by an operator, all four events are sent to Vitria: one createAccount event, one orderHeader event, one DSL addService event, and one DSL addProduct event. The events will be processed by the Order Processor Model and finally the Service Process Model. These models handle the interaction between the OMS provisioning system 610 and billing systems 604, 606. A provisioningRequest event is generated from this order as well. The provisioningRequest event is then sent to the OMS provisioning system 610.

[0068] Upon receiving the request, the provisioning system 610 will send back a provisioningResponses for confirmation. As a result of the response event, Vitria will send an updateProductStatus event to the Siebel CRM 602 in order to set the status of the product to 'Pending.'

[0069] Once provisioning is complete, the OMS provisioning system 610 will send two serviceStatusNotification events to Vitria denoting the DSL service and it's product is now 'Active.' Vitria will then send a serviceStatusNotification event to the Siebel CRM 602 denoting that the DSL service is active. Vitria will also send an

updateProductStatus event to the Siebel CRM 602 to change the status of the DSL product to 'Active.' At the same time, the addservice event and addproduct event will be sent to one or more of the billing systems 604, 606. The orderHeader event is internal to Vitria and will not be sent to billing.

[0070] The channel architecture 600 described above is a configuration of Vitria's native channel architecture. The integration hub has defined an account, product, new order, and billing channels to which business events are published. The account, product, and new order channels are routed to the composite billing channel. The account, product, and new order channels are prioritized to ensure that business events are sent to billing and provisioning in the correct order.

[0071] Various implementations of the systems and techniques described here may be realized in digital electronic circuitry, integrated circuitry, specially designed ASICs (application specific integrated circuits), in computer hardware, firmware, software, or combinations thereof, or in other types of machines.

[0072] A number of embodiments of the present invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. Accordingly, other embodiments are within the scope of the following claims.